Genetic Programming Search for $\Lambda_c^+ \rightarrow pK^+\pi^-$ and $D_s^+ \rightarrow K^+K^+\pi^-$

PANIC 2005

24 October 2005

Eric W. Vaandering

ewv@fnal.gov

Vanderbilt University

for

FOCUS Collaboration

Overview

- Introduction to Genetic Programming
  - Populations and Generations
  - Mutation and Crossover
  - Fitness and Natural Selection
- Genetic Programming applied to the doubly Cabibbo suppressed decays $\Lambda_c^+ \to pK^+\pi^-$ and $D_s^+ \to K^+K^+\pi^-$
- Exploration of GP specific systematic uncertainties
- Conclusions

What is Genetic Programming

Genetic programming is a machine learning algorithm based on two assumptions:

To find solutions to a problem, we mimic biology and the evolutionary process.

Since we will use computer programs to implement our solutions, the *form* of our solution should *be* a computer program.

Genetic Programming applies a biological model which includes reproduction, mutation, and survival of the fittest to automatically discover computer programs.

- Pioneered by John Koza: *Genetic Programming: On the Programming of Computers by Natural Selection* (1992)
- Since 1992, more than 3,000 papers applied to a wide range of problems in many disciplines

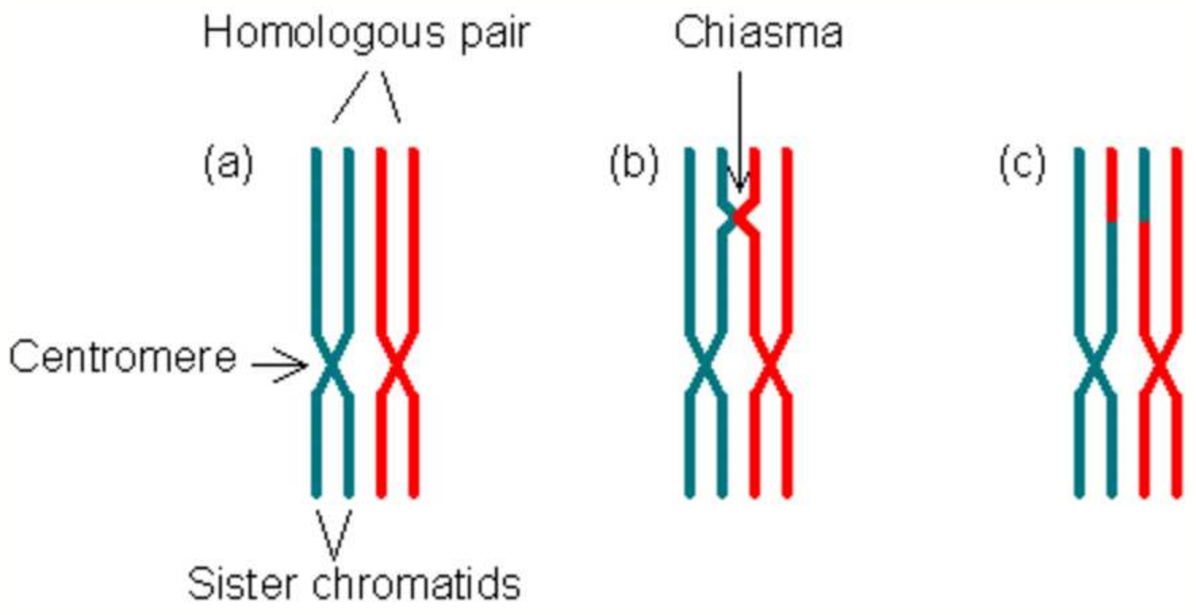
Populations and Generations

Genetic Programming works by transforming one group of programs (simple event filters in our case) in generation n into another group of programs in generation $n + 1$. There are typically a few hundred to a few thousand programs per generation.

The initial programs in the 0^{th} generation are generated completely randomly.

Gene Cross-over and Mutation

1) Biological (DNA) Cross-over



2) Mutations in nature change the genetic code for a small region of DNA. Usually are harmful or neutral; occasionally helpful (creates a better/different protein).

Mutations can restore lost (or never present) diversity.

These two processes, combined with natural selection, drive biological evolution.

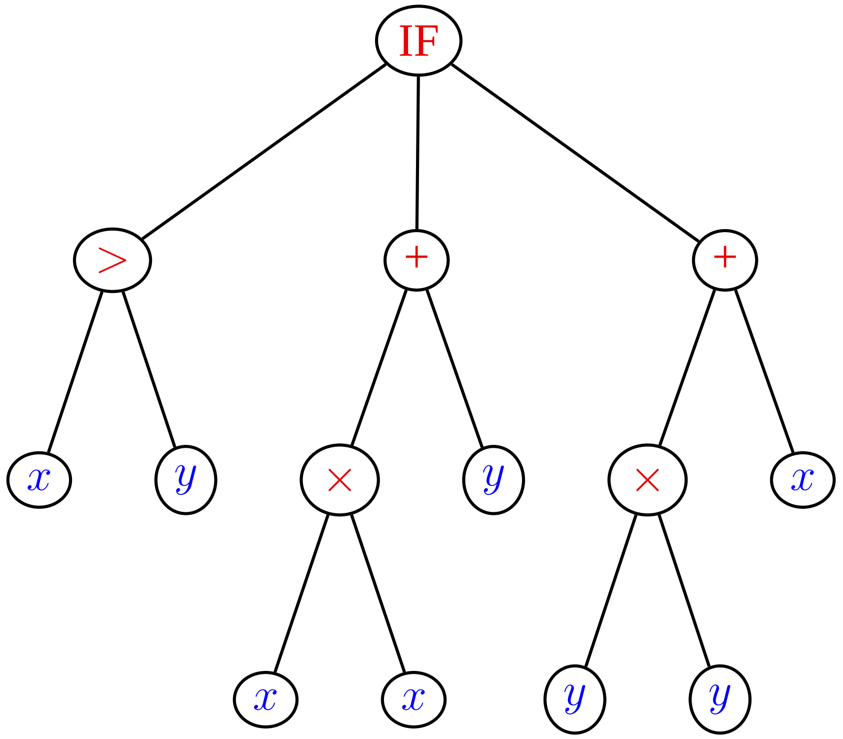
Tree Representation

Genetic Programming fundamentals are easier to illustrate if we adopt a “Tree” representation of a program. An example of this representation:

Code:

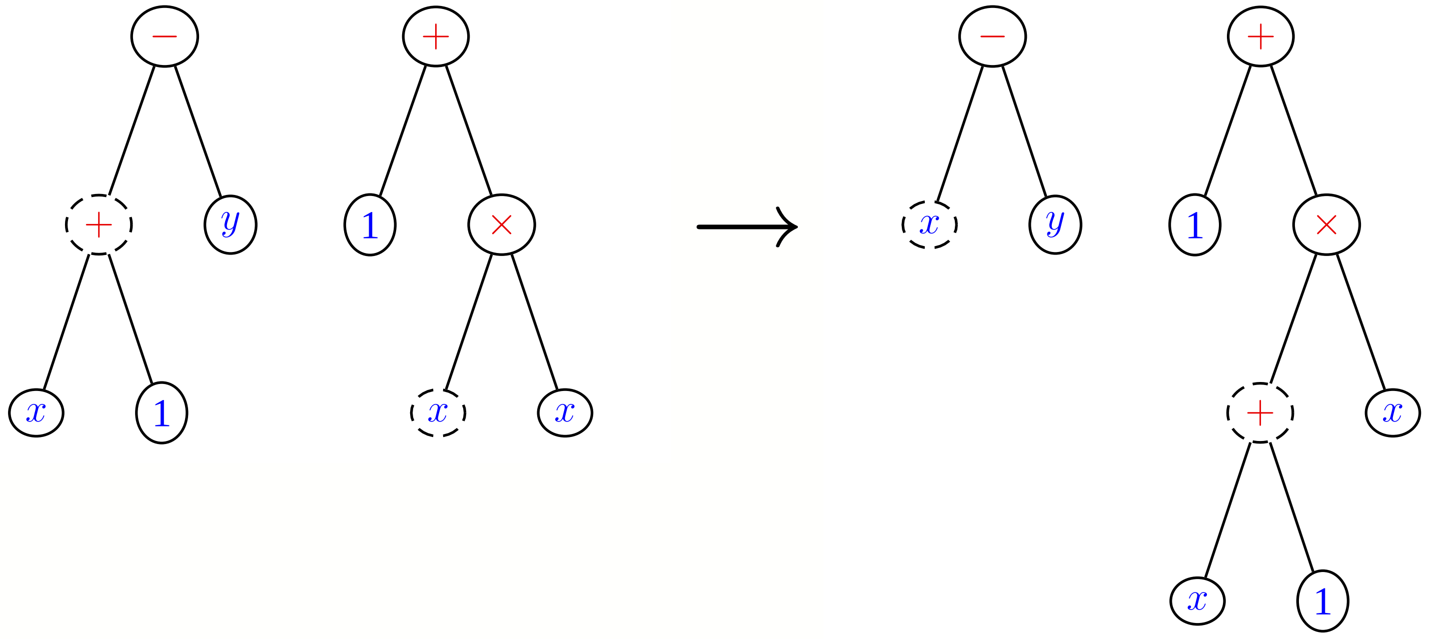
```
float myfunc(float x, float y) {
    float val;
    if (x > y) {
        val = x*x + y;
    } else {
        val = y*y + x;
    }
    return val;
}
```

Program tree



Crossover (Recombination)

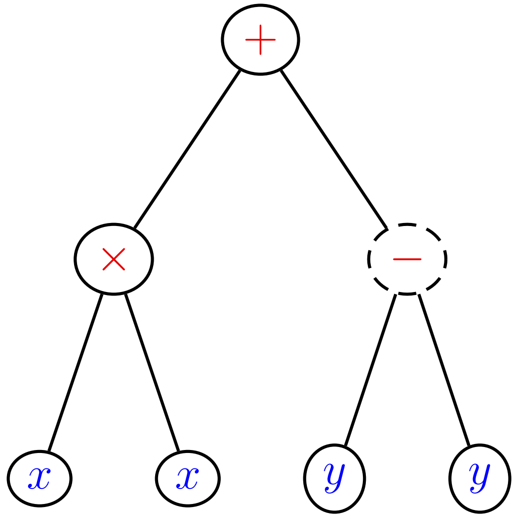
Two programs and crossover points within them are chosen. Sub-trees are removed and swapped between trees, giving two new “children”



It may combine the best aspects of both parents into one child (of course, we are just as likely to end up with the worst aspects in one child).

Mutation

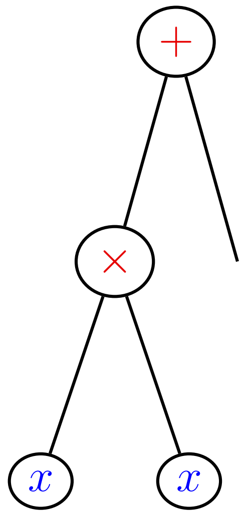
Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point

Mutation

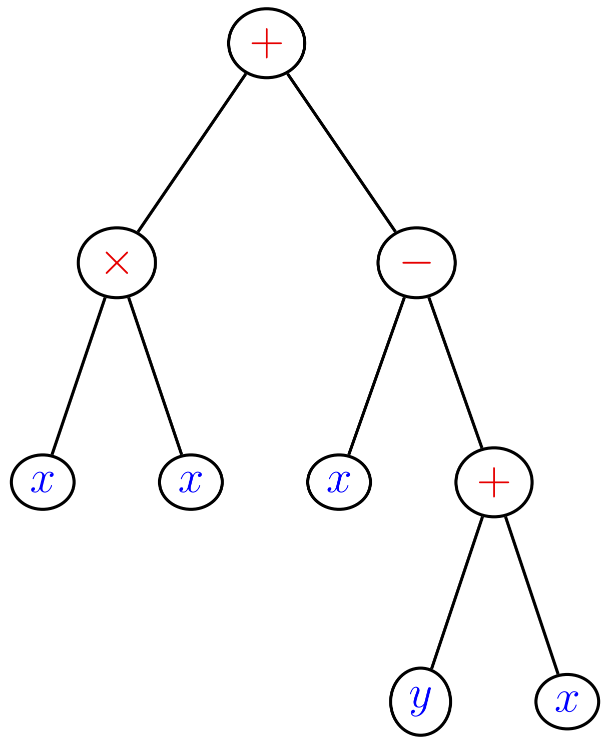
Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point
Remove the sub-tree

Mutation

Occasionally we want to introduce a mutation into a program or tree.



Pick a parent & mutation point
Remove the sub-tree
Randomly build a new sub-tree

Mutation can often be very destructive in Genetic Programming
Remember, both crossover and mutation are random processes.

Survival of the Fittest

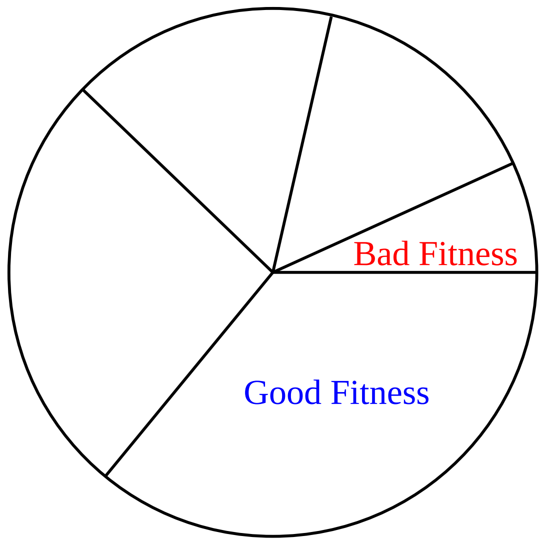
In nature, we know that the more fit an organism is for it’s environment, the more likely it is to reproduce. This is one of the basic tenets of evolutionary theory.

The Genetic Programming method mimics this by determining a *fitness* for each individual. Which individuals reproduce is based on that fitness.

- The better the fitness, the better the solution
- The problem *must* allow for inexact solutions. There may be a single *correct* solution, but there must be a way to distinguish between increasingly incorrect solutions.

Reproduction Probabilities

To select which individuals are chosen to populate the next generation, they are randomly chosen according to their fitness. We use something like a roulette wheel where the size of the slot is proportional to the fitness.



- The best individual is *most likely* to be chosen, but is *not guaranteed* to be chosen
- The worst individual *may* be chosen
- Best “sub-programs” (like genes) tend to survive

Running the GP

Putting it all together, we are ready to “run” the GP (find a solution).

- User has defined functions, variables, and measure of fitness
- Generate a population of programs (few hundred to few thousand) to be tested
- Test each program against fitness definition
- Choose genetic operation (crossover/mutation) and individuals to create next generation, randomly according to fitness
- Repeat process for next generation
  - Often tens of generations are needed to find the best solution
- At the end, we have a large number of solutions; look at the best

Application to HEP

OK, so all this is interesting to computer scientists, but how does it apply to physics, specifically HEP?

In FOCUS, we typically select interesting (signal, we hope) events from background processes using cuts on interesting variables. That is, we construct variables *we* think are interesting, and then require that an event pass the AND of a set of selection criteria.

Instead, what if we give a Genetic Programming framework some variables we think might be interesting, and allow *it* to construct a filter for the events?

- If an AND of cuts is the best solution, the GP can find that

Questions

When considering an approach like this, some questions naturally arise:

- Does it do as well as normal cut methods do?
- How do we know it’s not biased?
- The tree can grow large with useless information.
- Is it evolving or randomly hitting on good combinations?

Evaluating the GP

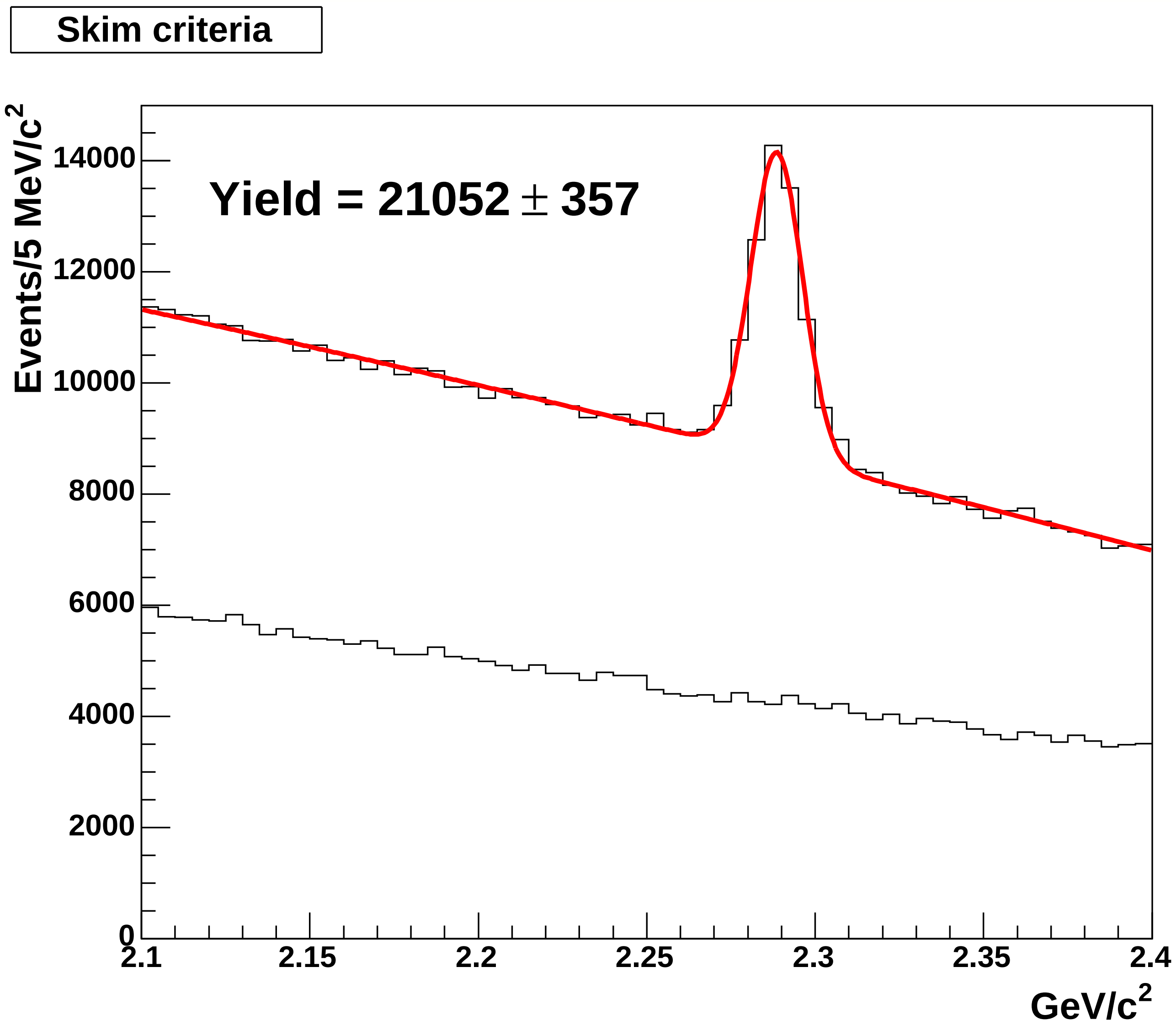
We chose to work with doubly Cabibbo suppressed decays normalized to Cabibbo favored decays where only the charge of the particles is different. Two nearly identical decays $\longrightarrow$ less stringent systematics.

We use the GP generated trees to classify each event as "signal-like" or "background–like." We do this for both CF and DCS decays and only consider "signal-like" events.

We then calculate a fitness related to the projected significance from fits to the CF signal and DCS background with the signal region blinded.

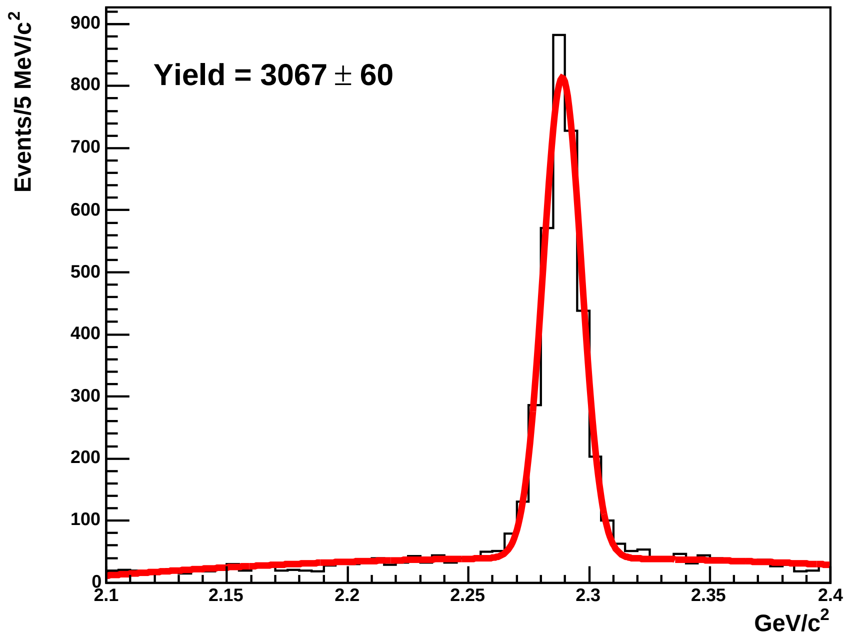
The GP optimizes this fitness as described.

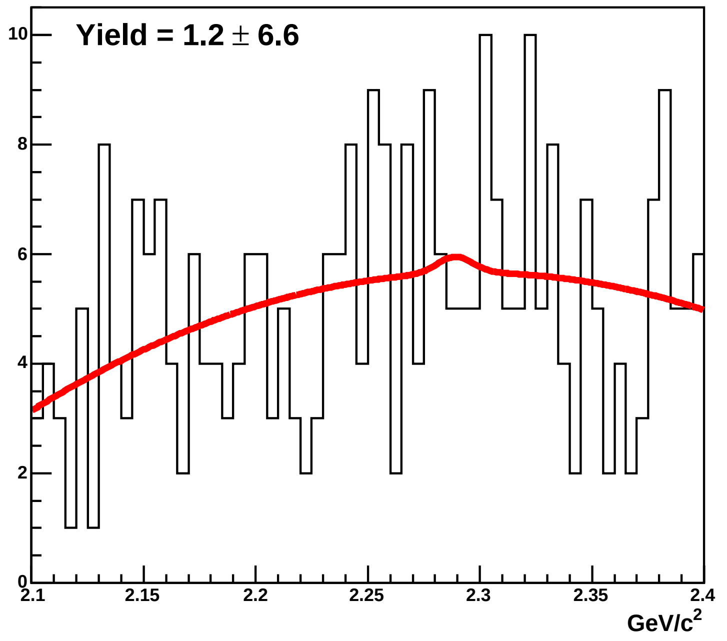
Pre-GP selection signals



Lower histogram: $\Lambda_c^+ \rightarrow pK^+\pi^-$ candidates

Signals after GP selection

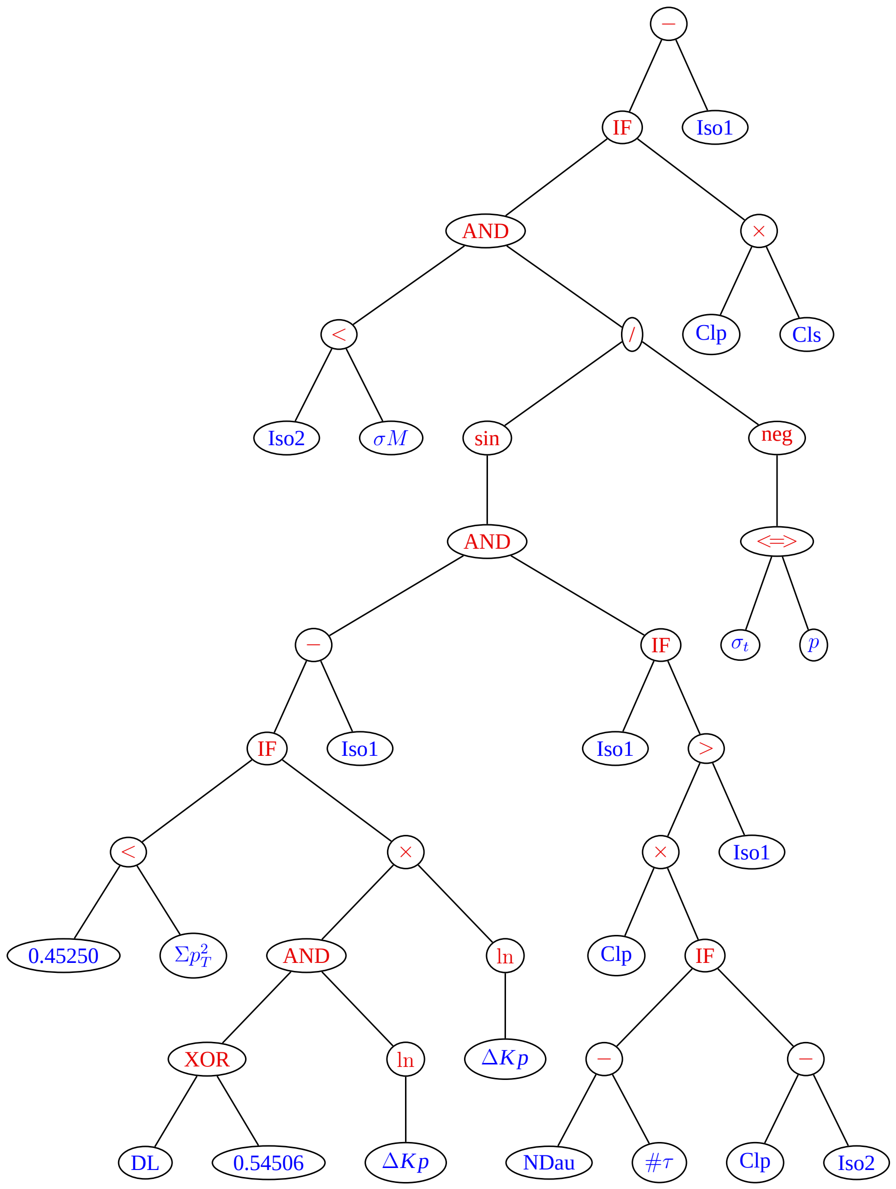




GP retains 3,000 of 21,000 original CF events
DCS background reduced $1000\times$

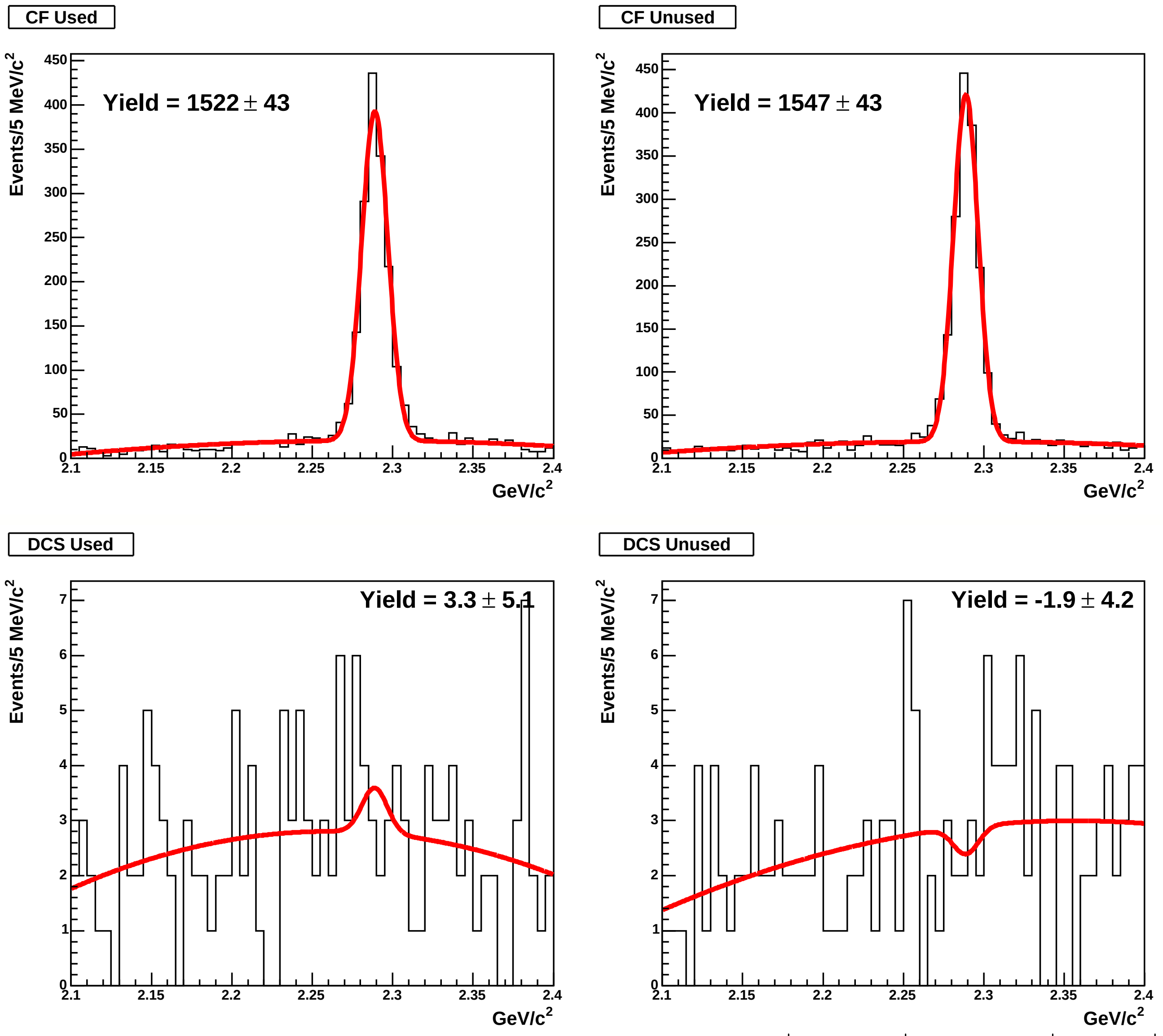
Fits during optimization use 1st degree polynomial.
Analysis uses 2nd degree

Best tree (72^{nd} generation)



GP Bias Check

Only optimized on half the events. Check the other half:



Data vs. MC comparisons

We picked DCS decays since the normalizing mode is nearly identical. It is important that the relative efficiency of the tree for CF and DCS modes is well modeled by our MC. Unfortunately, we can't measure this.

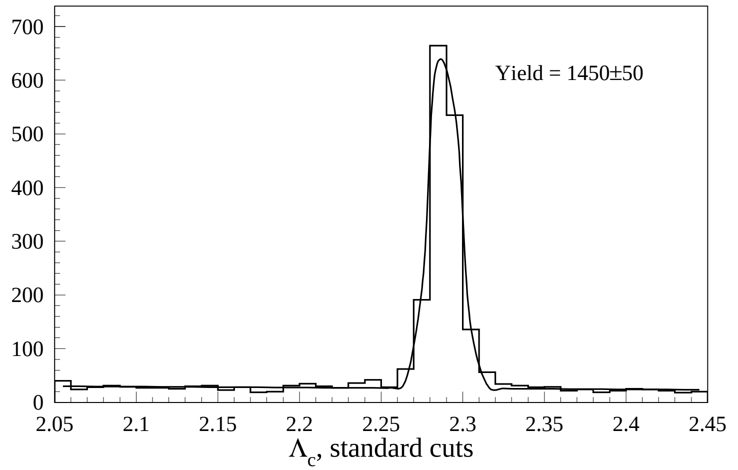
What we can measure is the efficiency of the tree (w.r.t. the base skim cuts) on data and MC:

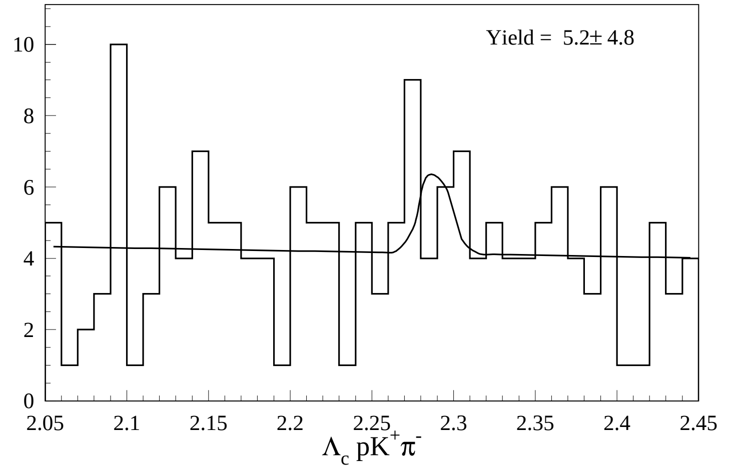
	Efficiency (GP/Skim)
Data	$(14.5 \pm 0.4)\%$
Monte Carlo	$(14.9 \pm 0.1)\%$

We always see such excellent agreement, which means that these multi-dimensional selections are well modeled by our MC. We take the nominal discrepancy (0.4%) as a systematic error.

Comparison with Cut Method

Compare this analysis with an old attempt with normal method:



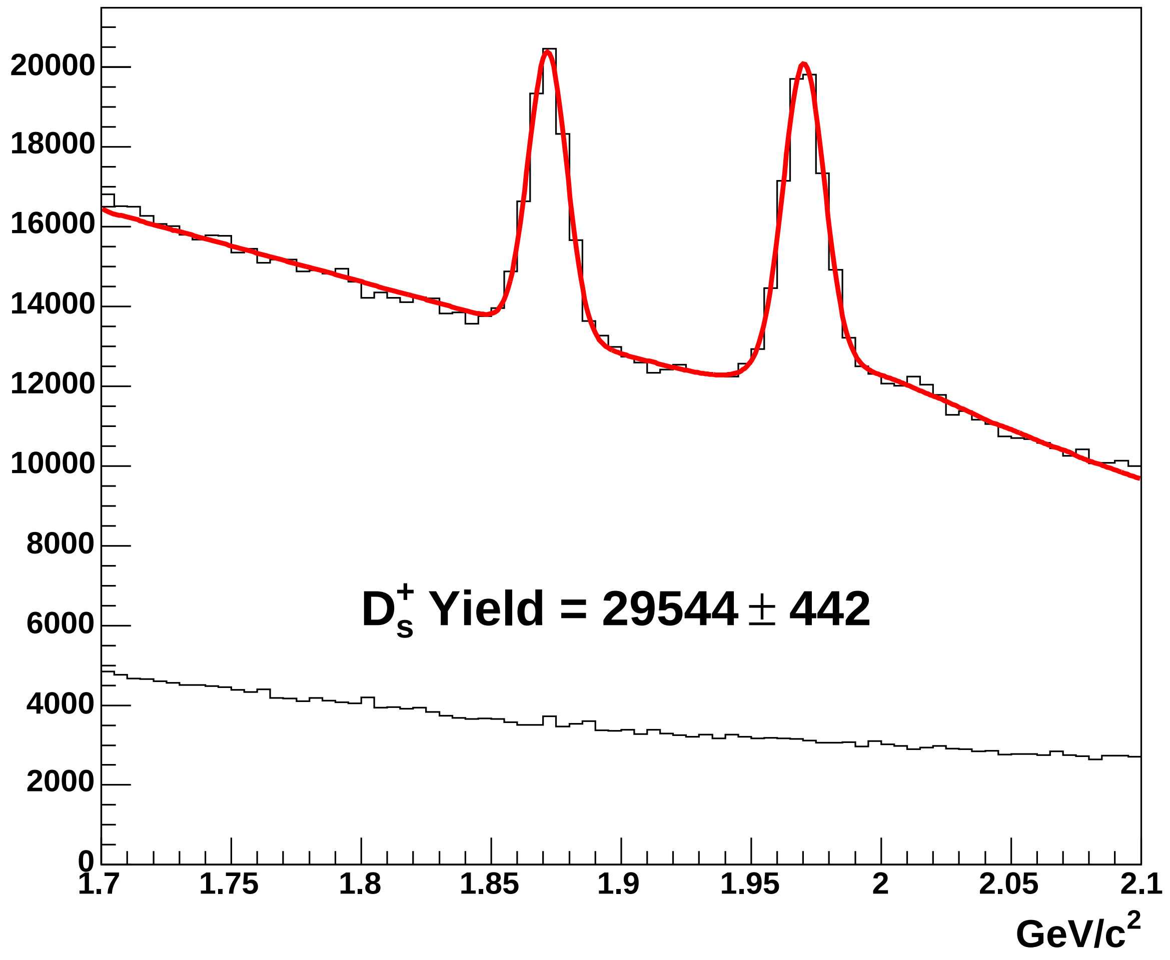


Branching Ratio with GP:
$0.05 \pm 0.26\%$
Branching Ratio with cuts:
$0.36 \pm 0.33\%$

Studies with $D^+ \to K^+\pi^+\pi^-$ show $\approx 2\times$ more signal at same signal/noise. (Shown in NIM article.)

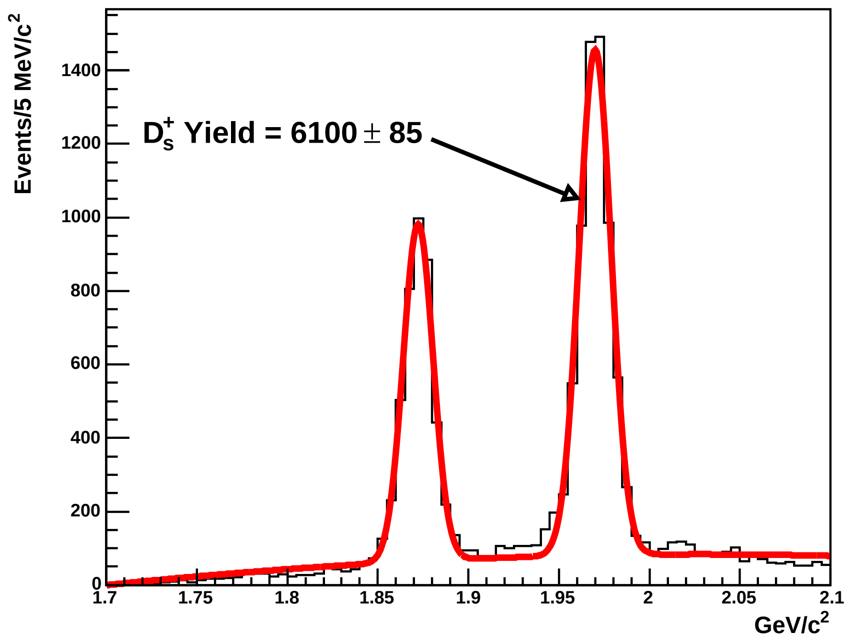
D_s^+ Initial Sample

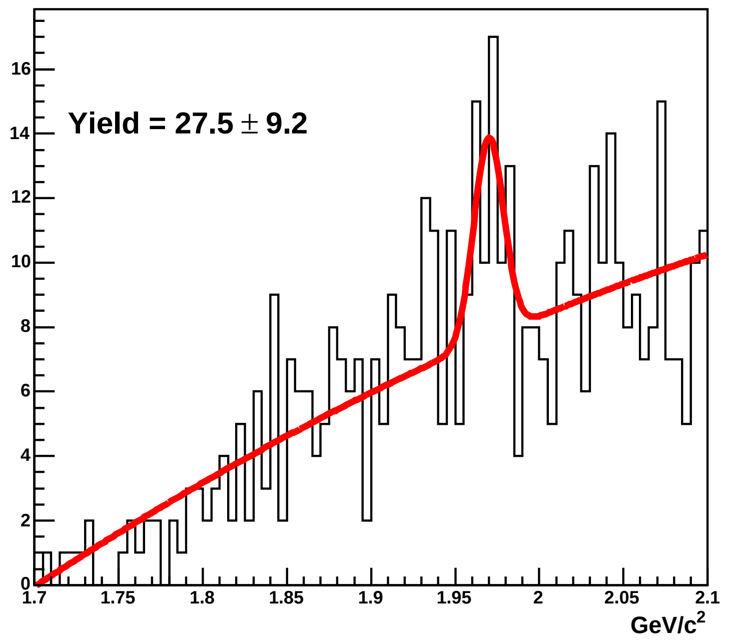
Contributions from $D^+ \rightarrow K^-\pi^+\pi^+$ (mis-ID) and Cabibbo suppressed decay $D^+ \rightarrow K^-K^+\pi^+$ included.



Signals after GP selection

$D^+ \to K^-\pi^+\pi^+$ (mis-ID) events removed





GP retains 21% of original events, DCS BG reduced $\sim 500\times$

Optimization uses 1st degree polynomial, analysis 2nd deg.
No real difference (27.9 ± 9.3 events for 1st deg.)

Summary of measurements

We also apply more traditional physics systematics studies to get percentage uncertainties on our knowledge of the relative (DCS/CF) uncertainties:

	Syst. Unc. (%)	
Source	Λ_c^+	D_s^+
MC statistics	0.6	0.4
DCS resonances	5.3	10.7
CF resonances	2.1	2.6
GP fi lter	2.6	3.5
Total	6.3	11.6

which gives us the following central values and limits:

Decay mode	Central Value	Limit (90% CL)
$\frac{\mathrm{BR}(\Lambda_c^+ \to pK^+\pi^-)}{\mathrm{BR}(\Lambda_c^+ \to pK^-\pi^+)}$	$(0.05 \pm 0.26 \pm 0.02)\%$	$< 0.46\%$
$\frac{\mathrm{BR}(D_s^+ \to K^+K^+\pi^-)}{\mathrm{BR}(D_s^+ \to K^-K^+\pi^+)}$	$(0.52 \pm 0.17 \pm 0.11)\%$	$< 0.78\%$

Conclusions

- I hope, in this very short time, I've given you a flavor of what Genetic Programming is
- These are the first limits for DCS decays of D_s^+ and Λ_c^+
- We have published two articles on this subject
  - NIM article on method: hep-ex/0503007, NIMA 551, pg. 504
  - D_s^+ and Λ_c^+ rel. branching ratio results: hep-ex/0507103, PLB 624, pg. 166
- We have shown that GP can be used in HEP event selection (this is the first application on HEP data)
- Can be used to improve sensitivity over traditional techniques
- Efficiency of GP event selection is well modeled in FOCUS

We think this novel method deserves further exploration

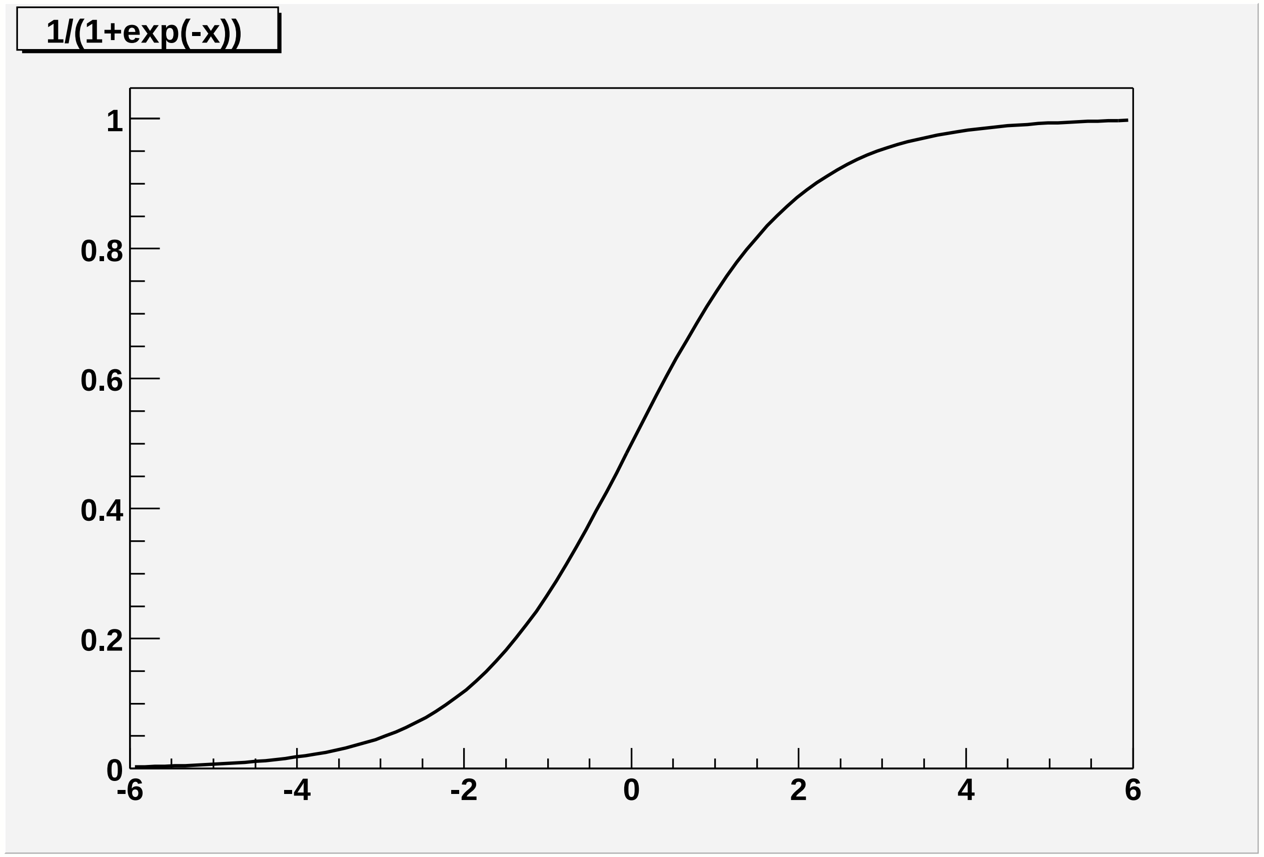
Backup slides

Backup slides

$f(n)$ function

A threshold function used in neural networks:

$$f(n) = \frac{1}{1+e^{-n}}$$

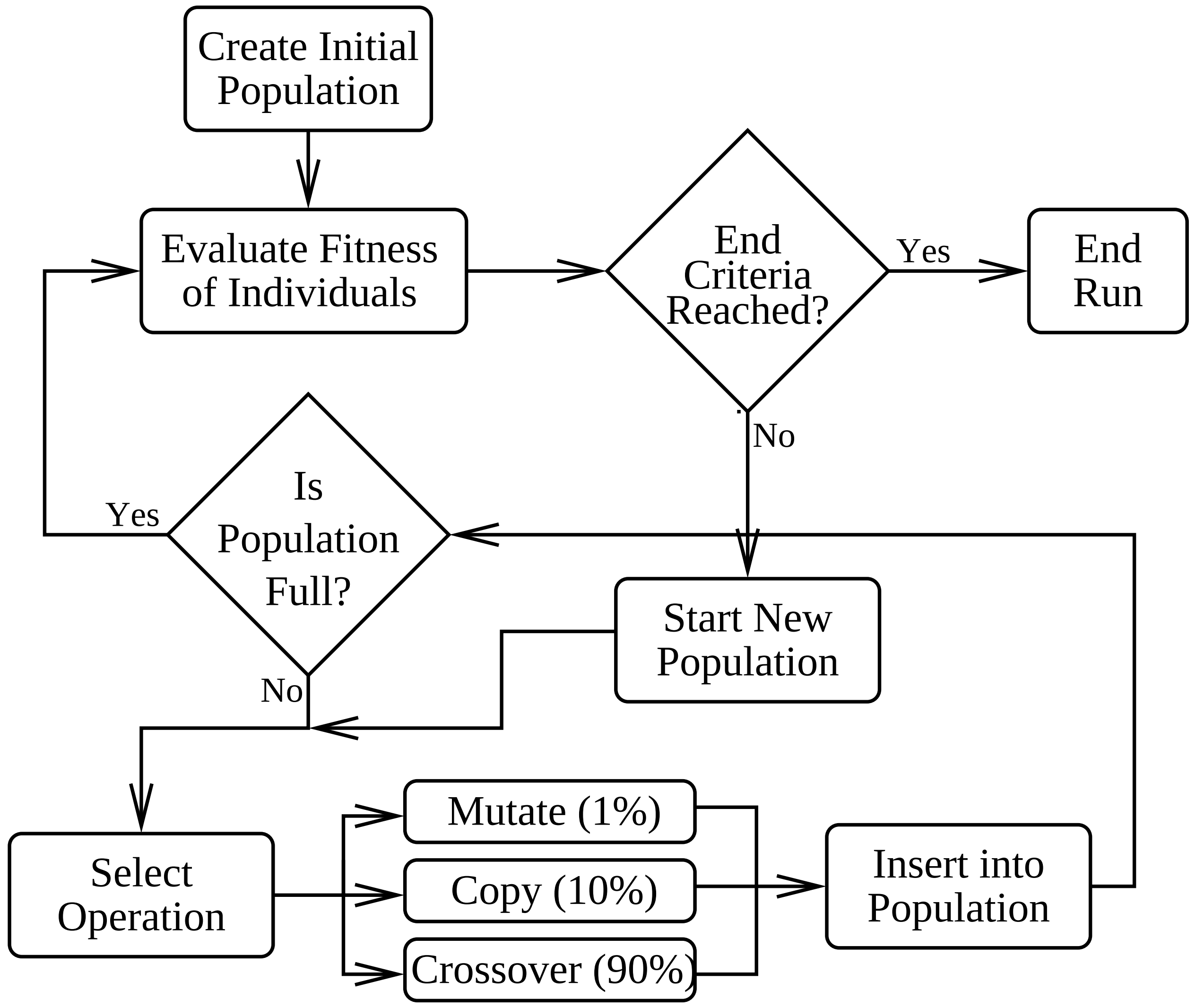


SW Mechanics & Conclusions

Is interfacing to an existing experiment's code difficult?

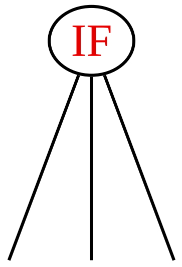
- Genetic programming framework
  - C language based **lilgp** from MSU Garage group
  - Modified for parallel use (**PVM**) by Vanderbilt Med Center group
  - Parallel version allows sub-population exchange
- Physics variables start with standard FOCUS analysis
  - Write **HBOOK** ntuples, convert to Root Trees
  - Write a little C++ code to access Trees, fill and fit histograms (using **MINUIT**) and return the fit information to the lilgp framework
- This is actually pretty easy

Genetic Programming Process



Building a tree

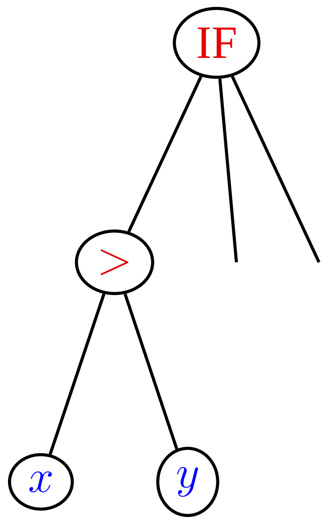
Trees are randomly built up one node at a time.



Root node ’IF’ has 3 args.

Building a tree

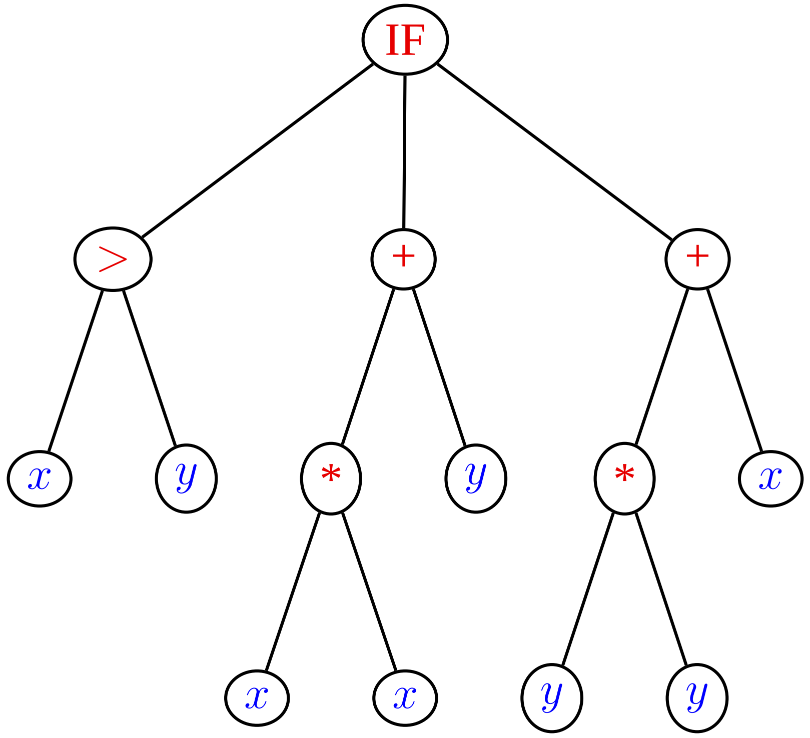
Trees are randomly built up one node at a time.



Root node 'IF' has 3 args.
'$>$' chosen for 1st arg.
x and y terminate '$>$'

Building a tree

Trees are randomly built up one node at a time.



Root node 'IF' has 3 args.
'$>$' chosen for 1st arg.
x and y terminate '$>$'
Remaining branches grown
Tree is complete
(all branches terminated)

Parallelizing the GP

Each test takes a while (10–60 sec on a 2 GHz P4) so spread over multiple computers

- Adopt a South Pacific island type model
  - A population on each island (CPU)
  - Every few generations, migrate the best individuals from each island to each other island
- Lots of parameters to be tweaked, like size of programs, probabilities of reproduction methods, exchanges, etc.
  - None of them seem to matter all that much, process is robust

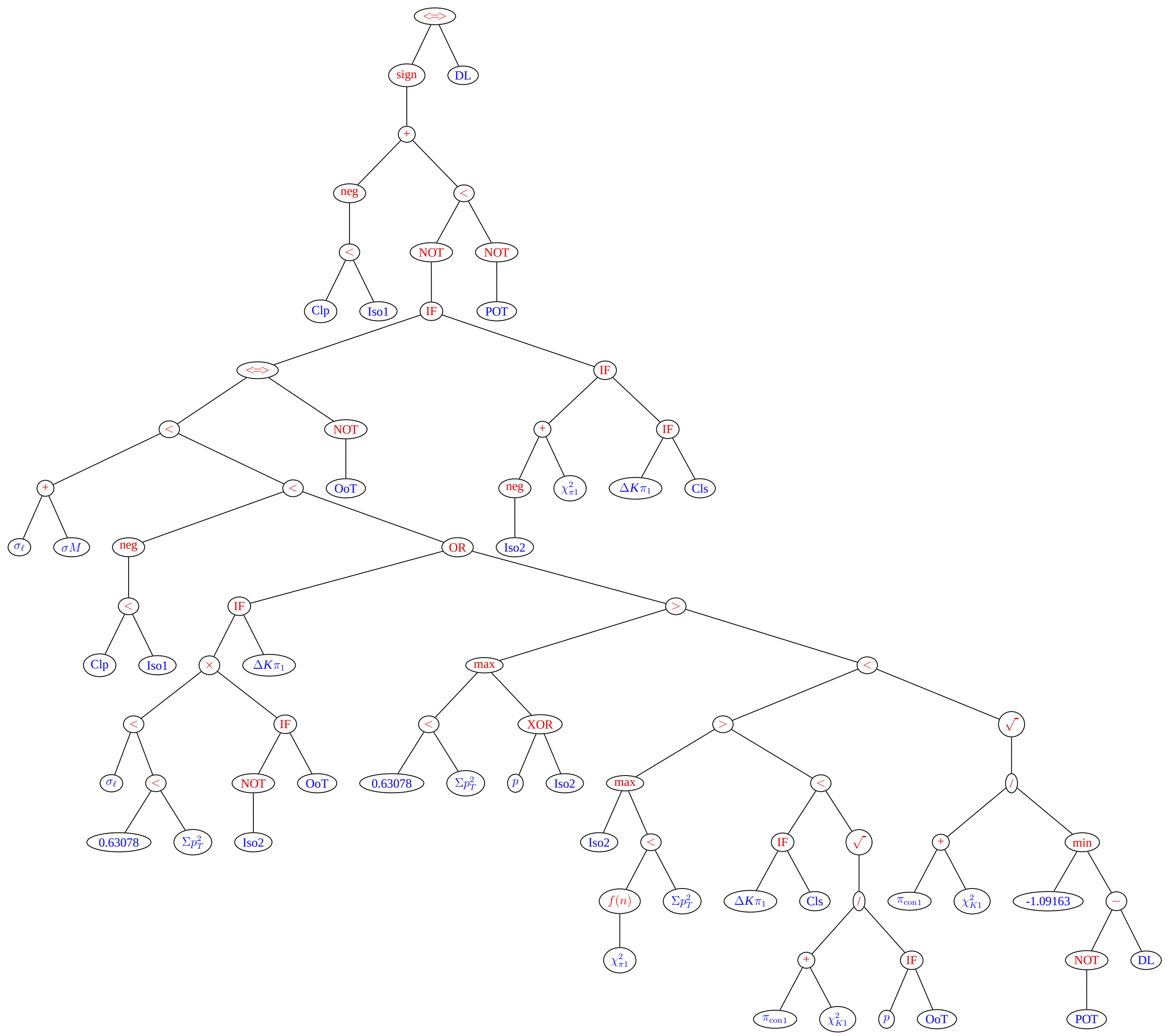
Practical considerations

Obviously, a tree can grow nearly infinite in size. This is usually undesirable. There are ways to control this:

- Set limits on number of nodes
- Set limits on depth of nodes
- Create initial topologies of specified depth

A common approach is to allow half of the initial population to grow completely randomly and to create the other half at a range of (shallow) depths. In the latter case, pick functions for all nodes $<$ desired depth, pick terminals for all nodes at desired depth.

Best D_s^+ tree (63rd generation)



<=>
sign
DL
+
neg
<
<
NOT
NOT
Clp
Iso1
IF
POT
<=>
IF
<
NOT
+
IF
+
<
OoT
neg
$\chi^2_{\pi 1}$
$\Delta K\pi_1$
Cls
σ_ℓ
σM
neg
OR
Iso2
<
IF
>
Clp
Iso1
×
$\Delta K\pi_1$
max
<
<
IF
<
XOR
>
$\sqrt{}$
σ_ℓ
<
NOT
OoT
0.63078
Σp_T^2
p
Iso2
max
<
/
0.63078
Σp_T^2
Iso2
Iso2
<
IF
$\sqrt{}$
+
min
$f(n)$
Σp_T^2
$\Delta K\pi_1$
Cls
/
$\pi_{\text{con}1}$
χ^2_{K1}
-1.09163
−
$\chi^2_{\pi 1}$
+
IF
NOT
DL
$\pi_{\text{con}1}$
χ^2_{K1}
p
OoT
POT